Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Chelsea Finn, Pieter Abbeel, Sergey Levine

Presented by Shih-Ming Wang

10-30-2019

Authors

Chelsea Finn
Assistant Professor
CS, Stanford University

Pieter Abbeel
Professor
EECS, UC Berkeley

Sergey Levine
Assistant Professor
EECS, UC Berkeley

Introduction I

Meta-Learning on Wiki

1. Traditional machine learning is based on a set of assumptions about the data, its "inductive bias", which means a learning algorithm will work well if the bias matches the learning problem.
2. By using different kinds of meta data, like properties of the learning problem, algorithm properties (like performance measures), or patterns previously derived from the data, it is possible to learn, select, alter or combine different learning algorithms to effectively solve a given learning problem.

Introduction II

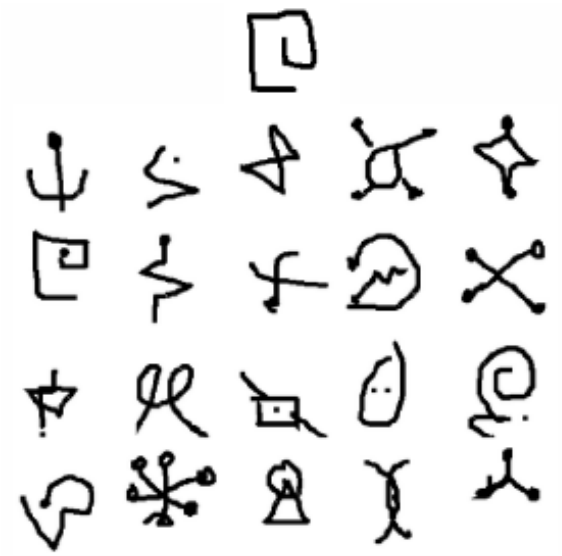
Machine Learning 101

1. The error of a machine learning algorithm is composed of two terms
   1. bias error: simple model can't capture the true data distribution
   2. variance error: complex model can capture the true data distribution but requires more data to reduce the possibility of overfit.
2. Usually, the model complexity is decided before we see the data, otherwise, we are overfitting the data by our brain
3. Therefore, meta-learning is a technique that use meta-data to dynamically change the model's inductive bias to a strong bias (low variance) that capture the true data distribution.

Introduction III

Few-Shot Learning Problem

1. Requires Meta-Learning to generalize well
2. Example problem: 1-shot, 20-way classification

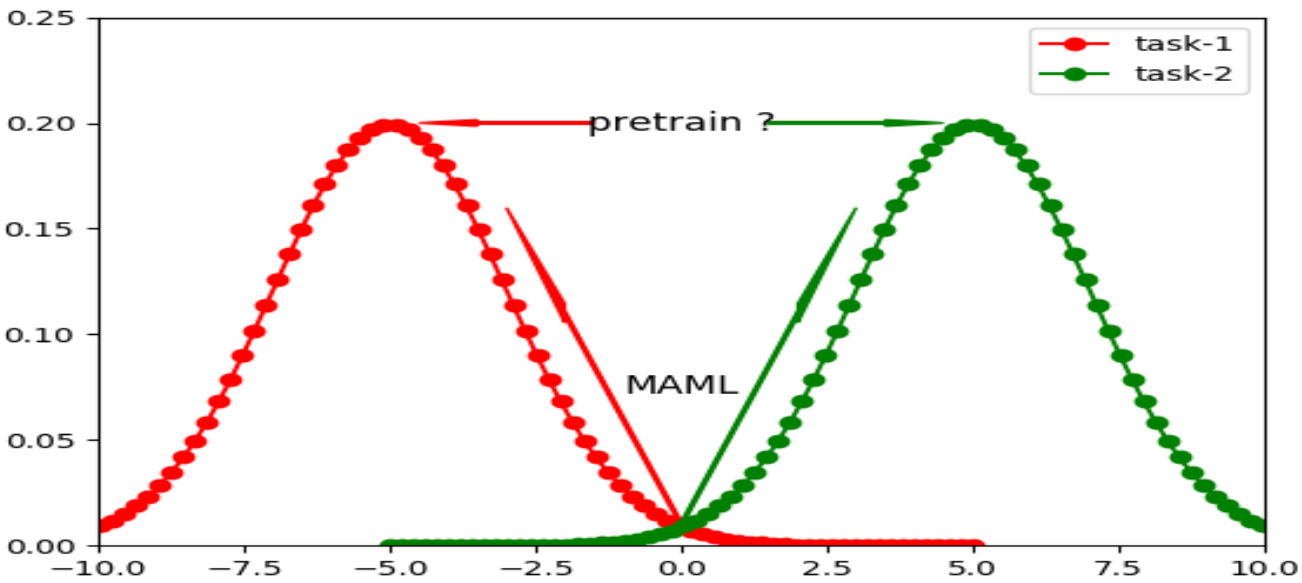


Introduction IV

MAML

1. Previous approaches to few-shot learning are usually specific to certain problems or model architecture
2. This paper proposed a meta-learning algorithm (MAML) that is
   1. general to different problem - supervised learning, reinforcement learning
   2. agnostic to different model architectures - CNN, RNN

MAML In a Nutshell



1. MAML searches for a point maximizing improvement of k-step fine-tuning for all tasks
2. Pre-training searches for a point maximizing performance on pre-training dataset

Notations

A formal & general definition of Model and Task

1. Model: $f : \mathcal{X} \to \mathcal{A}$
2. Task: $\mathcal{T} = \{H, \mathcal{L}(x_1, a_1, \cdots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t)\}$
   1. Episode Length: H
   2. Loss function: $\mathcal{L} : \mathcal{X}^{\mathcal{H}} \times \mathcal{A}^H \to \mathcal{R}$
   3. Initial observation distribution: $q(x_1)$
   4. Transition distribution: $q(x_{t+1}|x_t, a_t)$

Examples

	Supervise Learning	Reinforcement Learning
$\mathcal{X}$	images	states
$\mathcal{A}$	classification	action
H	1	arbitrary
$\mathcal{L}$	categorical cross-entropy	negative reward

Meta-Learning Algorithm I

1. The meta-learner maintain the model parameter θ
2. The task searcher perform SGD to find better $\theta \rightarrow \theta_i'$ on each task $\mathcal{T}_i$
3. Update θ according to the **loss on** θ_i' summing over all taks

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Meta-Learning Algorithm II

1. The task searching update $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ can easily be generalized to K updates with K samples per task.
2. MAML doesn't overfit easily because the meta-learner's update θ on loss $\mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ instead of $\mathcal{L}_{\mathcal{T}_i}(f_\theta)$
3. This makes that even one sample for each task enough

Meta-Learning Algorithm III

Computation Expense

Note that in meta-learner's update (using θ in place of f_θ)

$$\begin{aligned}
&\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta_i') \\
=&\nabla_\theta \theta_i' \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(\theta_i') && \text{(chain rule)} \\
=&\nabla_\theta \left[\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta)\right] \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(\theta_i') && \text{(task update)} \\
=&\left[I - \alpha \nabla_\theta^2 \mathcal{L}_{\mathcal{T}_i}(\theta))\right] \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(\theta_i')
\end{aligned}$$

we need to calculate the hessian-vector product $\nabla^2 \mathcal{L}_{\mathcal{T}_i}(\theta) \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(\theta_i')$, when omitted, $\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(\theta_i')$ is the first order approximation to the meta-learner's gradient $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta_i')$.

Meta-Learning Algorithm IV

1. Algorithm 1 can be adapted to different tasks including supervised learning & reinforcement learning by choosing different task loss function $\mathcal{L}_{\mathcal{T}_i}$ and data sample procedure
2. For example, in reinforcement learning:
   1. The model f_ϕ is a policy network that maps from states $x_t \in \mathcal{X}$ to a distribution over actions $a_t \in \mathcal{A}$ at each time step $t \in \{1, \cdots, H\}$.
   2. Given the initial state x_0, the model samples a trajectory $(x_0, a_0, x_1, a_1, \cdots, x_H)$. In K-shot reinforcement learning, the model is limited to sample only K trajectories.
   3. The loss function is the negative expectation of the total reward

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{x_t, a_t \sim f_\phi, q_{\mathcal{T}_i}} \left[\sum_{i=1}^{H} R_i(x_t, a_t) \right]$$

* This loss is generally not differentiable w.r.t. ϕ. For more information, please check this blog post and this slide

Meta-Learning Algorithm V

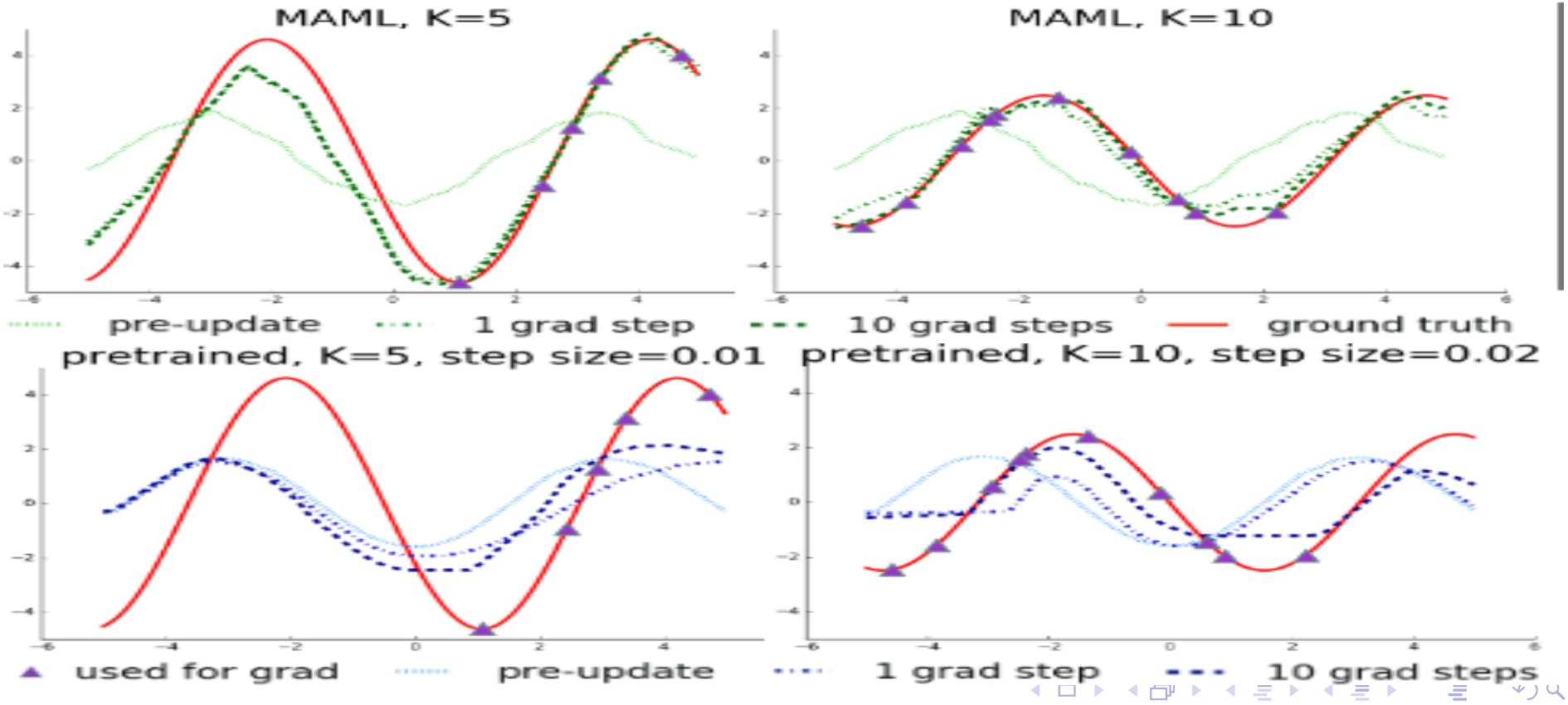
1. Algorithm 1 can then be adapted as Algorithm 3
2. Note that we sample from $\mathcal{T}_i$ two times for task update (using current θ) and meta-learner update (using θ_i).

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

Sinusoid Regression

1. Sin functions with different amplitude ranging $[0.1, 5.0]$
2. Each task (sin functions) has $K = 10$ samples
3. The task searcher adopt single step

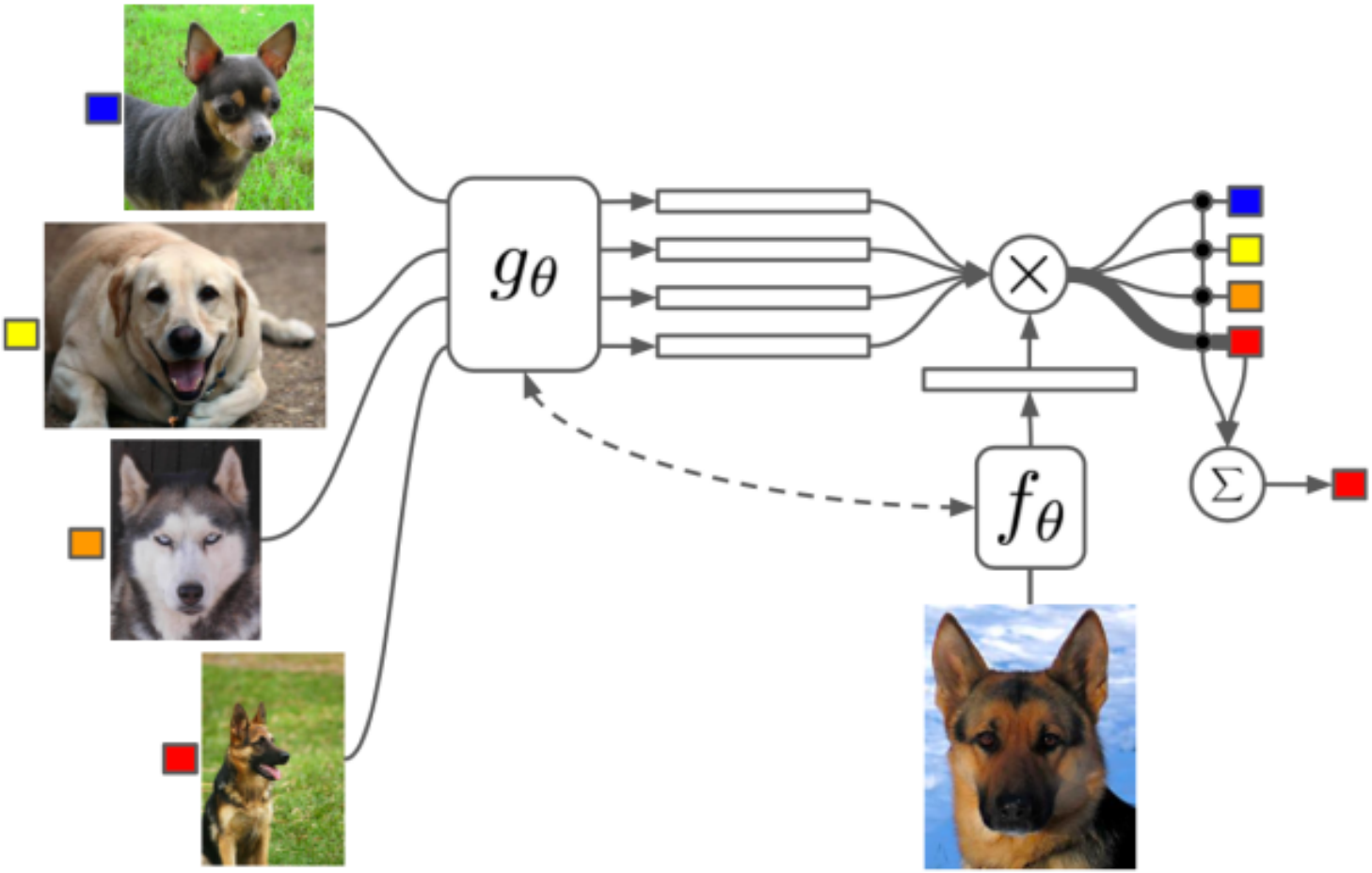


Classification

1. 5-way, K-shot image classification on MiniImagenet
2. Siamese nets, matching nets, and the memory module approaches are all specific to classification, and are not directly applicable to regression or RL scenarios.

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy	
	1-shot	5-shot
fine-tuning baseline	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
nearest neighbor baseline	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
matching nets (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
meta-learner LSTM (Ravi & Larochelle, 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx. (ours)	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML (ours)	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

Matching Net



Reinforcement Learning

1. Locomotion: agent needs to learn to run in a particular direction or at a particular velocity.
2. Each task means running in different directions or velocities.